Annexe 6. Notions d'ordonnancement.

APP3 Optimisation Combinatoire:
problèmes sur-contraints et ordonnancement.
Mines-Nantes, option GIPAD, 2011-2012.

Sophie.Demassey@mines-nantes.fr

Résumé

Ce document rappelle quelques définitions de base en ordonnancement et présente différents problèmes d'ordonnancement parmi les plus courants.

1 Définitions

> *Ordonnancer c'est programmer l'exécution d'une réalisation en attribuant des ressources aux tâches et en fixant leurs dates d'exécution.* [Carlier&Chrétienne, 1988]

Tout problème d'ordonnancement se définit donc par un ensemble de *tâches*, encore appelées *activités*, à exécuter sur un ensemble de *ressources*. Il s'agit d'affecter, à chaque activité, une date d'exécution et un sous-ensemble de ressources de façon à satisfaire à la fois des *contraintes temporelles* (par exemple, respecter un ordre partiel sur l'ensemble des activités) et des *contraintes d'accès aux ressources* (par exemple, exécuter sur une ressource au plus une activité à chaque instant). Enfin, la programmation des activités doit généralement répondre à un *critère d'optimisation* (par exemple, minimiser la durée totale de l'ordonnancement). Il existe une grande variété de problèmes d'ordonnancement ; chacun étant défini par une combinaison spécifique de caractéristiques des données, des contraintes et de l'objectif.

1.1 Données

Activités. En ordonnancement statique déterministe, l'ensemble $A = \{a_1, \ldots, a_n\}$ des activités est a priori connu, de même que, pour chaque activité a_i, sa *durée* $p_i \in \mathbb{Z}^+$ et le nombre et la quantité, appelée *consommation*, de ressources requises pour son exécution. Souvent, on précise également pour chaque activité a_i, une *date échue* ou *deadline* d_i ; une activité est alors dite *en retard* si elle termine son exécution après cette date.

Un problème est dit *préemptif* si les activités sont morcelables – leur exécution est complète mais interruptible à plusieurs reprises – et *non-préemptif* dans le cas contraire.

Ressources. Une ressource est définie par sa *capacité* : la quantité de la ressource disponible à chaque instant pour l'exécution des activités. Cette capacité est soit fixe, soit variable dans le temps : une période d'indisponibilité est par exemple représentée par une capacité nulle sur la période. Une ressource dite *consommable* (par exemple, un budget ou une matière première) a une capacité nécessairement variable : la quantité de ressource allouée à une activité n'est plus disponible au terme de l'exécution de l'activité. Au contraire, une ressource *renouvelable* (par exemple, une machine, une personne, un processeur) a généralement une capacité fixe : la quantité de ressource est allouée à une activité provisoirement, le temps de son exécution. Deux types de ressources sont distinguées : les ressources *cumulatives* (par exemple, un four, un groupe de personnes) peuvent exécuter plusieurs activités simultanément (dans la limite de leur capacité), tandis que les ressources *disjonctives* (par exemple, un processeur, une personne) ne peuvent exécuter qu'une seule activité à la fois : la capacité des ressources et la consommation des activités sont alors fixées à 1.

1.2 Contraintes

Contraintes temporelles. Elles sont de deux types : les premières définissent les dates d'exécutions possibles ou interdites pour chaque activité ; les secondes définissent les séquencements possibles ou interdits entre activités. Ces dernières contraignent alors la date d'exécution d'une activité relativement à la date d'exécution d'une autre activité ou d'un sous-ensemble d'activités. Les contraintes temporelles les plus fréquentes sont les suivantes :

fenêtres de temps : à chaque activité a_i est associée une *fenêtre de temps* $[r_i, d_i]$ dans laquelle l'activité doit obligatoirement et intégralement être exécutée. r_i est la *date de disponibilité* et d_i la *date d'achèvement* de l'activité a_i.

disjonctions : des ensembles d'activités sont mutuellement incompatibles : au plus une activité de l'ensemble peut être en cours d'exécution à un instant donné.

précédences simples : un ordre partiel $(A, <)$ entre les activités doit être respecté par l'ordonnancement final : $a_i < a_j$ indique que l'activité a_i *précède* l'activité a_j. Autrement dit, l'exécution de a_j ne peut commencer avant la fin de l'exécution de l'activité a_i.

précédences généralisées : un graphe orienté valué $G = (A, E, t)$, appelé *graphe potentiels-tâches*, est spécifié. S'il existe un arc $(a_i, a_j) \in E$ valué par $t_{ij} \in \mathbb{R}$, alors la date de début de a_j est supérieure à la date de début de a_i plus la valeur t_{ij} dans tout ordonnancement.

Le graphe potentiels-tâches permet d'exprimer différentes contraintes temporelles, y compris le séquencement relatif des activités, les fenêtres de temps, les temps de set-up. On remarque en effet que les précédences simples $(A, <)$ sont un cas particulier des précédences généralisées, car elles peuvent être représentées par le graphe (A, E, t) avec $(a_i, a_j) \in E$ si $a_i < a_j$ et $t_{ij} = p_i$. De plus, les précédences simples permettent de définir des fenêtres de temps, en ajoutant deux activités fictives a_0, précédant toutes les activités, et a_{n+1}, que toutes les activités précèdent, puis en calculant r_i la valeur du plus long chemin dans le graphe de a_0 à a_i, et d_i la valeur du plus long chemin de a_i à a_{n+1}. Enfin, les disjonctions sont le plus souvent considérées comme des contraintes de ressources : à chaque ensemble d'activités mutuellement incompatibles, peut être associée une ressource disjonctive fictive.

Contraintes de ressources. la contrainte commune à tout problème d'ordonnancement est la suivante : la somme des consommations des activités en cours d'exécution à un temps t et allouées à une même ressource r ne doit pas excéder la capacité de la ressource à cet instant.

Des contraintes propres aux problèmes multi-ressources, précisent l'affectation des ressources aux activités. D'une part, selon les problèmes, une activité devra s'exécuter sur une seule ressource ou bien sur plusieurs ressources simultanément. D'autre part, le sous-ensemble des ressources affecté à chaque activité peut être connu (donnée d'entrée) ou bien à décider. Dans les problèmes multi-ressources cumulatifs, l'affectation des ressources aux activités est généralement pré-définie : on connaît la consommation exacte de chaque activité sur chacune des ressources. Inversement, dans les problèmes multi-ressources disjonctifs, aussi appelés *problèmes à m-machines parallèles*, l'affectation des ressources aux activités (*allocation de ressources*) doit être décidée au même titre que l'affection des dates d'exécution (*ordonnancement*). Dans ce cas, si les ressources sont identiques, alors on connaît pour chaque activité le nombre de ressources nécessaires à l'exécution de l'activité. Si les ressources ne sont pas identiques, alors la durée d'une activité peut varier selon la ressource qui lui est affectée. Enfin, dans les problèmes à m-machines parallèles préemptifs, les morceaux de tâches peuvent être allouées à des machines différentes (nécessairement à des instants différents).

Ordonnancements réalisables. Un *ordonnancement* est généralement représenté par le vecteur des dates de début des activités $(S_1, \ldots, S_n)$. Dans les problèmes avec allocation de ressources, une solution réalisable doit être caractérisée par un second vecteur des ressources affectées aux activités. On note :

- S_i la *date de début* d'exécution de l'activité a_i ;
- C_i la *date de fin* d'exécution de a_i : $C_i = S_i + p_i$;
- C_{max} la date de fin de l'ordonnancement, appelée *makespan* : $C_{max} = \max_i C_i$;
- L_i ou T_i le retard de a_i : $L_i = \max\{0, C_i - d_i\}$;
- U_i l'indicateur de retard de a_i : $U_i = 1$ si $C_i > d_i$, $U_i = 0$ sinon ;

Pour être réalisable, l'ordonnancement doit satisfaire l'ensemble des contraintes temporelles et contraintes de ressources du problème. Lorsqu'il s'agit d'un problème de satisfaction, on précise généralement parmi les contraintes temporelles, une date de fin au plus tard de l'ordonnancement T, appelée l'*horizon*. La date de début d'ordonnancement est trivialement fixée à 0.

1.3 Objectif

Ordonnancements optimaux. Les problèmes d'ordonnancement sont le plus souvent traités comme des problèmes d'optimisation. Les critères d'optimisation suivants sont les plus usuels :

durée totale : minimiser le makespan C_{max} ;

valeur des retards : mimimiser le plus grand retard $L_{max} = \max_i L_i$, ou bien la somme pondérée des retards $\sum_i w_i L_i$;

nombre de retards : mimimiser le nombre (pondéré) des retards $\sum_i (w_i) U_i$;

nombre d'interruptions : dans les problèmes préemptifs uniquement, minimiser le nombre total d'interruptions des activités ou bien le nombre maximal d'activités interrompues.

durée d'interruptions : dans les problèmes préemptifs uniquement, le *flow time* $C_i - S_i$ d'une tâche a_i indique la durée de son interruption égale à $C_i - S_i - p_i$; minimiser le flow time maximal ou la somme pondérée des flow times.

2 Problèmes d'ordonnancement

De nombreuses combinaisons possibles des paramètres décrivant un problème d'ordonnancement ont été étudiées : problèmes disjonctifs/cumulatifs, préemptifs/non-préemptifs, avec une/plusieurs ressources, avec durées unitaires/identiques/arbitraires, etc. La complexité de ces problèmes varie en fonction de ces différents paramètres, depuis les problèmes résolubles en temps linéaire (1 machine sans contraintes temporelles) jusqu'aux problèmes les plus fortement combinatoires, dont les instances les plus petites sont impossibles à résoudre à l'optimum en temps raisonnable.

Cette section décrit quelques classes génériques de problèmes d'ordonnancement parmi les plus fréquents. Les problèmes sont notés $\alpha|\beta|\gamma$, suivant le schéma de classification de (Graham, Lawler, Lenstra and Rinnooy Kan, 1979). Ces trois paramètres décrivent respectivement les ressources (α), les caractéristiques des activités (β), et le critère d'optimisation (γ).

Problème central de l'ordonnancement.
- notation : $-|prec|Cmax$
- entrée : un ensemble A de tâches non-préemptibles ; une durée $p_i > 0$ pour chaque tâche a_i ; pas de contraintes de ressources (autrement dit, une ressource cumulative de capacité infinie) ; un graphe potentiels-tâches sans cycle (de coût positif).
- sortie : un ordonnancement réalisable minimisant $Cmax$.
- solutions : deux solutions particulières sont calculables par un algorithme de plus long chemins dans le graphe potentiels-tâches : l'ordonnancement au plus tôt ($S_i = l(0,i)$, où $l(i,j)$ dénote la longueur du plus long chemin de a_i à a_j dans le graphe) et l'ordonnancement au plus tard ($S_i = l(0,n+1) - l(i,n+1)$). L'ordonnancement au plus tôt est optimal. En l'absence de cycle, le problème central de l'ordonnancement appartient donc à la classe de complexité $\mathcal{P}$.

Problèmes à 1 machine.
- exemple : $1|r_i, d_i|Cmax$
- entrée : un ensemble A de tâches non-préemptibles ; une ressource renouvelable disjonctive ; une durée $p_i > 0$ pour chaque tâche a_i ; fenêtres de temps.
- sortie : un ordonnancement réalisable minimisant $Cmax$.
- relaxation : ce problème est $\mathcal{NP}$-difficile au sens fort, mais il est facile à évaluer en pratique en considérant sa relaxation préemptive. L'ordonnancement préemptif de Jackson (JPS) consiste à ordonnancer une tâche a_i de plus petite date échue d_i aussitôt que celle-ci devient disponible

($t = r_i$), interrompant éventuellement par là-même une tâche a_j moins prioritaire ($d_i < d_j$) précédemment ordonnancée ($r_j < t$). Cet ordonnancement est optimal pour le problème à 1 machine préemptif $1|r_i, d_i, pmtn|Cmax$ et il se calcule en temps polynomial $O(n \log n)$. Cette borne joue un rôle central dans la résolution des problèmes d'ordonnancement disjonctif difficiles, tels que les problèmes d'atelier.

Problèmes préemptifs à m machines parallèles identiques.

- exemple : $Pm|r_i, pmtn| \sum w_i U_i$
- entrée : un ensemble A de tâches préemptibles ; une durée $p_i > 0$ et un coefficient $w_i \geqslant 0$ pour chaque tâche a_i ; m ressources disjonctives (machines) ; chaque tâche s'exécute sur au plus une machine par instant (consommation $b_i = 1$) ; les morceaux d'une même tâche peuvent s'exécuter sur des machines différentes ; fenêtres de temps.
- sortie : une allocation des ressources aux (morceaux de) tâches et un ordonnancement réalisables minimisant la somme pondérée des indicateurs de retard $\sum_i w_i U_i$.
- solutions : les problèmes préemptifs peuvent souvent être modélisés comme des problèmes d'affectation (des unités de tâches aux instants) ou de flot dans un graphe de taille pseudo-polynomiale (fonction de l'horizon T d'ordonnancement). Ces problèmes sont alors, dans le pire des cas, $\mathcal{NP}$-difficiles au sens faible. En pratique, il peut exister des algorithmes spécifiques de programmation dynamique de complexité moindre, voire des algorithmes polynomiaux pour les résoudre. Dans l'exemple ci-dessus, un ordonnancement réalisable correspond à un flot de valeur $\sum_i p_i$ dans le réseau de transport $G = (N, A)$ défini sur l'ensemble des sommets $N = \{s, p\} \cup A \cup \{t_1, \ldots, t_T\}$. À chaque tâche a_i, est associé un arc (s, a_i) de capacité p_i. À chaque instant $t = t_1, \ldots, t_T$, est associé un arc (a_i, t) de capacité 1 pour chaque tâche a_i telle que $t > r_i$ et un arc (t, p) de capacité m. Pour le critère donné, un ordonnancement optimal correspond à un flot de coût minimal dans le graphe G légèrement modifié : tout sommet a_i est dédoublé en a_i^0 et a_i^1 et on ajoute deux arcs de capacité p_i : (a_i, a_i^0) de coût 0 et (a_i, a_i^1) de coût w_i. Les arcs (a_i, t) sont alors partagés en deux groupes : (a_i^0, t) si $t \leqslant d_i$ et (a_i^1, t) si $t > d_i$. Le calcul d'un tel flot n'est pas efficace et il existe pour ce problème un algorithme pseudo-polynomial plus efficace en pratique (voir Baptise, 2002). Cet algorithme s'exécute même en temps polynomial quand les durées des tâches sont toutes identiques, i.e. $Pm|p_i = p, r_i, pmtn| \sum w_i U_i$ appartient à $\mathcal{P}$. Par ailleurs, le critère du $Cmax$ est plus facile que le nombre de retards : le problème à m machines préemptif $Pm|r_i, pmtn|Cmax$ appartient à $\mathcal{P}$, tandis que sa version non-préemptive $Pm|r_i|Cmax$ se résoud en temps pseudo-polynomial. À noter que les problèmes d'ordonnancement sont souvent plus faciles dans leur version préemptive que dans leur version non-préemptive, mais ce n'est pas toujours le cas. Par exemple, quand le nombre de machine n'est pas fixé et que toutes les durées sont égales, le problème préemptif $P|p_i = p, pmtn| \sum w_i U_i$ est $\mathcal{NP}$-difficile, tandis que la version non-préemptive $P|p_i = p| \sum w_i U_i$ est dans $\mathcal{P}$.

Problèmes d'atelier. Les tâches, appelées *opérations* dans ce contexte, sont regroupées en entités, appelées *travaux* ou *jobs*. Les ressources sont disjonctives et renouvelables et appelées *machines*. Un atelier contient m machines et chaque job est composé de m opérations affectée chacune à une machine différente. On note a_k^i l'opération du job J_i devant s'exécuter sur la machine M_k. Il n'y a pas de contraintes temporelles sur les jobs, sauf parfois des fenêtres de temps. En revanche, on distingue 3 types de contraintes temporelles liant les opérations d'un même job. Ils définissent 3 problèmes d'atelier différents :

flow-shop : les opérations d'un job sont liées par l'ordre total $a_1^i < \cdots < a_m^i$ pour tout job J_i ;

job-shop : les opérations d'un job sont liées par un ordre total connu, différent pour chaque job ;

open-shop : les opérations d'un job sont liées par un ordre total inconnu, i.e. elles sont en disjonction : deux opérations d'un même job ne peuvent être exécutées simultanément.

Ces problèmes sont pour la plupart $\mathcal{NP}$-difficiles au sens fort, à l'exception notable de certains problèmes d'atelier à 2 machines avec minimisation du makespan : $J2||Cmax$, en job-shop, est $\mathcal{NP}$-difficile au sens fort, mais $F2||Cmax$, en flow-shop, et $O2||Cmax$, en open-shop, appartiennent à $\mathcal{P}$.

Problème cumulatif (CuSP).

- notation : $PS1|r_i, d_i|-$
- entrée : un ensemble A d'activités non-préemptibles ; une ressource renouvelable cumulative de capacité B ; une durée $p_i > 0$ pour chaque activité a_i et une consommation $b_i \geqslant 0$ pour chaque activité a_i ; fenêtres de temps $[r_i, d_i]$ pour chaque activité a_i.
- sortie : un ordonnancement réalisable, i.e. tel que le sous-ensemble de tâches A_t ordonnancées à chaque instant t vérifie $\sum_{i \in A_t} b_i \leqslant B$ (ressource) et $a_i \in A_t \implies r_i \leqslant t < d_i - p_i$ (fenêtres de temps).
- complexité : ce problème de décision étend le problème à 1 machine ($B = b_i = 1$) et le problème à m machines ($b_i = 1$). Il est $\mathcal{NP}$-complet au sens fort. Une relaxation caractéristique du CuSP est basée sur la notion d'*énergie* qui correspond au produit de la quantité de ressource consommée et du temps. Dans le *CuSP élastique*, la consommation d'une tâche a_i est autorisée à varier dans le temps, mais l'énergie totale reste la même : $\sum_t b_{it} = p_i b_i$. Un algorithme de résolution de complexité temporelle $O(n^2)$ existe pour cette relaxation.

Problème d'ordonnancement de projet à contraintes de ressources (RCPSP).

- notation : $PS|prec|Cmax$
- entrée : un ensemble A d'activités non-préemptibles ; un ensemble R de ressources renouvelables cumulatives ; une durée $p_i > 0$ pour chaque activité a_i et une consommation $b_{ik} \geqslant 0$ pour chaque activité a_i et chaque ressource r_k ; une capacité $B_k > 0$ pour chaque ressource r_k ; contraintes de précédence simple.
- sortie : un ordonnancement réalisable minimisant $Cmax$.
- complexité : ce problème généralise tous les précédents. Il est naturellement $\mathcal{NP}$-difficile au sens fort. Plusieurs algorithmes approchés ont été proposés pour ce problème et obtiennent généralement de bonnes solutions. En revanche, les meilleurs algorithmes exacts (branch-and-bound) ne parviennent pas à fermer des petites instances (60 activités, 4 ressources) de la littérature.

3 Références

- *Problèmes d'ordonnancement : Modélisation / Complexité / Algorithmes*, J. CARLIER and Ph. CHRÉTIENNE, Masson, 1988.
- *Scheduling Algorithms*, P. BRUCKER, Springer Lehrbuch, 2001.
- *Résultats de complexité et programmation par contraintes pour l'ordonnancement*, Ph. BAPTISTE, HdR Université de Technologie de Compiègne, 2002.
- *Satisfiability tests and time-bound adjustments for cumulative scheduling problems*, Ph. BAPTISTE, C. LE PAPE, W. NUIJTEN, Annals of Operations Research 92 :305-333, 1999.
- *Optimization and approximation in deterministic sequencing and scheduling : a survey*, R. E. GRAHAM, E. L. LAWLER, J. K. LENSTRA and A. H. G RINNOOY KAN, Annals of Discrete Mathematics, 4 :287-326, 1979.
- *Complexity Results of Scheduling Problems*, P. BRUCKER and S. KNUST, http://www.mathematik.uni-osnabrueck.de/research/OR/class/.
- *The Scheduling Zoo*, C. DÜRR, http://www.lix.polytechnique.fr/~durr/query/.
- *CuSPLIB : A Library of Single-Machine Cumulative Scheduling Problems*, T. Yunes, http://moya.bus.miami.edu/~tallys/cusplib/.